To Appear: Artificial Neural Networks in Engineering (ANNIE) '91
November, St. Louis, MO

Learning Load-Balancing Strategies using Artificial Neural Networks

Pankaj Mehra and Benjamin W. Wah
University of Illinois at Urbana-Champaign

ABSTRACT

We describe SMALL, a program that learns load-balancing strategies for distributed multiprocessors. Load balancing exhibits ill-defined objective functions, delayed evaluative feedback, and non-Markovian state spaces, which makes learning strategies for load balancing a complex task. SMALL uses *comparator neural networks*, digital filters, and time-series analysis, for learning to compare objective function values of states resulting from a set of alternative moves. It employs a novel credit-assignment procedure suitable for load balancing and other non-Markovian learning tasks.

I. INTRODUCTION

We address automated learning of strategies for *load balancing* — the dynamic scheduling of independent jobs arriving at various sites of a network to minimize their completion time. Each job can be either executed locally or migrated to another (lightly loaded) computer for remote execution. Even with certain simplifying assumptions, finding the optimal schedule requires perfect knowledge of the future. In order to complete, a job must compete with other jobs for CPU, memory, and I/O. The completion time of a job depends in a complex and unknown fashion upon the level of load on various resources for which jobs contend. Optimization of completion time, the goal of load balancing, is not a well-formed function; its ill-posedness precludes prior specification of good strategies. Automated learning of load balancing strategies requires (i) learning more about the objective function; and, (ii) predicting future performance of jobs based on past values of observed workload.

We describe SMALL, a program for learning load balancing strategies. It evaluates each job as it completes, and communicates the evaluation to other sites. Learning programs at each site treat such evaluation as feedback for their (local) load-balancing strategy. Such strategies have two parts: (i) *decision metrics*, which contain status information about the load on that site; and, (ii) *decision policy*, rules for deciding where jobs should be executed. SMALL incorporates an experimentation environment that allows one to measure the completion time of a distributed batch of jobs under controlled loading conditions and different strategies. The measured performance is first translated into feedback (evaluation) and

Research supported by NASA Contracts NCC 2-481 and NAG-1-613.

then into new values for various parameters of load-balancing strategies (credit assignment). Strategy learning requires a rational solution to the problems of evaluation and credit assignment. We propose an alternative to the traditional credit assignment procedure formalized by Sutton et al. [3]. One can learn intelligent decision metrics that convey goal-related information. We draw upon previous work of Hwang et al. [2] in the area of performance evaluation and the work of Tesauro et al. [5] in the area of neural networks.

Computer workload is generated by processes, which compete for various resources of a system: CPU, network, memory, and disk. The load on a site is described by the level of utilization of its resources. These are highly dynamic and mutually independent quantities with significant autocorrelation. Since they characterize the current load on resources of contention, their values at the time of introducing a job influence its completion time. Other determinants of completion time, such as the size and frequency of service demands made by the job on various resources, are nearly impossible to estimate in practical situations. One can use the values of current load on various sites to compare the (projected) completion times, thus quantifying the benefits of migrating a task to a remote site.

Traditional load-balancing algorithms assume that: (i) contention for CPU is the key determinant of performance; (ii) a moving average of the number of processes in contention for CPU is a significant determinant of completion time; and, (iii) simple decision rules such as always sending a job to the least loaded site can be determined a priori and result in good performance. In reality, such decision metrics are only appropriate for purely computational tasks; their values cannot be compared in order to compare completion times of a job at various sites. Good decision procedures estimate the benefits as well as measure the costs of migration. Such information is usually installation-specific, and not available to an operating system designer. However, with experience, good metrics and rules can be 'learned' by observing the completion times under various loading conditions and scheduling policies. SMALL provides an environment for collecting and using such data to experimentally improve the rules and metrics used by existing load balancing policies.

II. THE STRATEGY LEARNING TASK FACING SMALL

Given a problem P, a problem solver S, and a learning environment E, a strategy learning system L uses the information provided by E to optimize various parameters of the strategy used by S to solve P. Various characteristics of P, S, and E affect the design of L.

In our case, P is the load balancing problem defined as follows: to minimize an objective function O, defined on the completion times of jobs, by using a parameterized decision process D(W) to decide where to execute each job. D(W) is a distributed decision process, each component of which schedules the jobs arriving locally at each site. denote by $D_M(W_M,L,a)$ the decision rule at a site M; $L(t)$ represents the load information available to M at time t; and, $a \in A$, where A is the set of possible decisions concerning a job arriving on site M at time t.

S contains the software for (i) managing the execution of jobs; (ii) measuring load-related parameters such as CPU utilization and network traffic; and, (iii) measuring goal-related information such as completion times of jobs. E reacts to the decisions made by S. First, it produces feedback for optimization of D(W) by evaluating the 'goodness' of load balancing. Second, it provides prior knowledge about the load-balancing problem, for instance, by providing models for the variation of $L(t)$ and the relationships between $L(t)$, D(W), and feedback.

The triple (P,S,E) constitutes a strategy learning task T. A learning system L uses F to compute changes ΔW to parameters of D such that future values of F will be more favorable. Conversion of F into ΔW, called the credit-assignment problem, can be divided into two stages: temporal credit assignment (TCA), wherein F is apportioned between recent decisions according to their eligibility; and, structural credit assignment (SCA), wherein the feedback per decision is apportioned between various parameters leading to that decision [1]. L consists of a critic C and a learning element LE, which are responsible for solving, respectively, the TCA and SCA problems.

T(P,S,E) poses complex problems of estimation and modeling. We use T_j^L and T_j^{NL} to denote, respectively, the completion time of a job j with and without load balancing. (In the latter case, all jobs are executed locally at their sites of origin.) O, the objective function, is to improve the value of $\Pr\{ \frac{T_j^{NL}}{T_j^L} > 1 \}$ for a random job j in future. O's dependence on completion time makes it ill-defined: without extensive models of jobs, computers, and background load, it cannot be written as a closed-form function of measurable parameters. Instead, O is a measured quantity. Such ill-posedness of load balancing precludes a priori knowledge of good strategies and decision metrics. L must use the measured values of completion time to empirically induce both goal-related metrics $F_M(L_M)$ and goal-achieving policies $D_M(W_M, F_M(L_M), a)$.

Measurements $L(t)$ are dynamic multi-dimensional quantities. The time spent waiting for resources depends upon the load. As the load on individual resources changes with time, so do the overall delays experienced by a job. Problems of inferring $F_M(L_M)$ therefore require (i) using recently observed loading patterns as a guide to their future variation; and, (ii) reducing multi-dimensional time-series information into scalar decision metrics which can be used for comparing

Violation of Markov property[1] can be illustrated by studying the autocorrelations of the $L(t)$ time series (Appendix). Nonzero partial autocorrelations at lags greater than one are exhibited by the load on all four resources. Consequently, past values affect future values of load given the current state. Markov property allows one to associate scalar evaluations with states. Its violation, forces us to attach evaluations to state sequences rather than to individual states. This complicates TCA immensely.

The feedback signal F evaluating a job j, produced when j finishes execution, is delayed with respect to decisions d about the placement of j. L must search for that direction and magnitude of ΔW which will yield favorable evaluation in

[1]*Markov property* asserts that future behavior of a system depends only on its current state.

future. Due to the job interaction and feedback delays, each instance of F evaluates several decisions, and each decision is eligible to receive a portion of several feedback signals. These facts complicate credit assignment. At equilibrium, jobs finish at least as fast as new jobs arrive. Therefore, feedback signals are produced at roughly the same rate as the overall arrival rate of jobs. This necessitates the design of efficient and intelligent procedures for credit assignment and history management.

III. THE SMALL SYSTEM

A. *Learning Decision Metrics*

A load balancing experiment involves several jobs. To combine the speed-ups of individual jobs into a scalar objective function for evaluating strategies we would like O to be of the form $\sum_j G(\frac{T_j^{NL}}{T_j^L})$, where $G(s) = 1 - e^{-c\frac{s-1}{s+1}}$. This form of O enables us to evaluate jobs independently, while simultaneously ensuring that (i) no job is slowed down too much; and, (ii) large speed-up on a few jobs cannot dominate slowdown over several others. At the time of scheduling, one can use predicted speedup as a parameter of decision making. This is exactly the quality we desire a decision metric to possess. However, because *absolute* completion times are not predictable with the information available at the time of making a decision, and because *relative* completion time is sufficient for choosing among alternatives, we formulate the following *prediction problem*:

> To learn functions $F_m(L_m)$ of local load so that $F_m(L_m) - F_n(L_n)$ approximates $\frac{T_j^m(L_m) - T_j^n(L_n)}{T_j^0(L_0)}$, where T_j^i represents the predicted completion time of a job j at machine i.

If a job j runs at site k with the least F_k, its predicted completion time relative to a fixed site 0 is the least of all sites considered. Assuming that the true completion time function permits linear ordering of (load, site characteristics) pairs, k is also the site where the completion time under load balancing will be the least. Normalization of $T_j^i(L_i)$ with $T_j^0(L_0)$ factors out the job-specific characteristics of completion time. We assume that waiting time grows linearly with job size. It ensures that (i) each F_i can be predicted using only information about load on site i; and, (ii) F_i values can be numerically compared for choosing the site with the least predicted completion time. Relative to the parameters W, the targets of the learning process, the completion time of a job under no load balancing (T_j^{NL}) is a constant quantity. Therefore, *for fixed values of parameters* W, F_i is a suitable decision metric for characterizing the current and past load on site i.

Estimation of F_i's entails (nonlinear) regression of T_j^i on time series $L_i(t)$. Autocorrelations in observed load are due to the fact that process population changes only as fast as the arrival rate of new jobs in the system. A job may take from 15 seconds to a few minutes to complete. Load values are typically sampled at a frequency of 10 to 100 Hz. Therefore, approximately 1000 past load values must be considered for modelling relative completion time. We decompose the modeling process into two stages: smoothing and fitting. Smoothing isolates the trend from the data and produces "filtered" data which can be sampled at lower frequencies. The lower sampling frequency results in smaller window sizes. Fitting models relative completion time as a function of (smaller) windows of current and past filtered load values.

B. Learning Decision Policies

Policy learning modifies the parameters W_m of D_m so as to maximize $\sum_j G(\cdot)$ for future jobs. Test data for this problem include jobs and load readings. Jobs are organized into streams, one per site, and load is generated using SMALL's artificial workload generation package described elsewhere. The test jobs are executed one with and once without load balancing. Each job's site of origin records its completion time in both cases. It then computes the G value, and broadcasts it on the network along with the job's identification number. These broadcasts become the feedback signal F.

When feedback upon completion of a certain job j is received, all decisions affecting its performance receive their share of its evaluation. Incremental assignment of credit is possible only because of the linear form of the objective function O with respect to completion-time values. Each site maintains — along with a history of recent decisions — information about the *causal and temporal scopes* of each decision. Briefly, causal scope is a binary value indicating whether the decision in question could have influenced the performance of job j whose performance evaluation is quantified by the current feedback signal F. For instance, if j was never run at a site m, and the sites at which j ran did not exchange any tasks with m while j was running, then decisions made at m are not eligible to receive any portion of F. Simple rules such as these constitue a causal model of E, the learning environment. Each decision also has a finite temporal scope, spanning the time axis from the start until the end of execution of the job being scheduled; it can only receive feedback from those jobs whose execution overlaps its temporal scope. In order to minimize the number of past decisions needing storage, decisions whose temporal scopes contain no active jobs can be 'forgotten.'

The critic, whose responsibility it is to perform TCA, computes an *eligibility* value by multiplying a binary causal parameter with a continuous-valued temporal parameter. The temporal parameter is based on a temporal model describing how eligibility of a decision varies with time elapsed since making the decision. It is reasonable in our case to assume maximum eligibility of 1 initially, and decaying (say, linearly) to 0 at the time of completion of the job being scheduled. The feedback signal is divided among the decisions in proportion to their eligibility.

Once a decision receives its share of the feedback signal, various parameters of the decision process that participated in making that decision are given their share of the feedback. This is the SCA problem and its solution depends on the specific representation of D(W). We interpret $D_m(W_m, F_m(L_m(t)), a)$ as the probability of choosing action a. Naturally, then, we would like to impose the constraint $\sum_{a \in A} D_m(W_m, F_m(L_m(t)), a) = 1$. Now, everytime the feedback signal is positive we increase the probability of the chosen action towards 1, while simultaneously driving the other probabilities towards 0. Similarly, when the feedback is negative, we can drive the probability of the chosen action towards 0 while pushing the other probabilities closer to 1. The modifications should be such that the constraint above is never violated. The process just described gives us $\Delta D_m(\cdot)$. Having reduced a problem of learning from evaluative feedback into one of learning from presecriptive feedback, we can now use error-correcting procedures similar to back-propagation [4] to compute ΔW_m.

APPENDIX: PARTIAL AUTOCORRELATION PLOTS FOR CPU LOAD

```
                        '.' MARKS TWO STANDARD ERRORS

LAG CORRELATION -1 9 8 7 6 5 4 3 2 1 0 1 2 3 4 5 6 7 8 9 1
  1     0.77239  |                  .|***************    |
  2     0.15422  |                  .|***                |
  3     0.12188  |                  .|**                 |
  4     0.07938  |                  .|**                 |
  5     0.04088  |                  .|*                  |
  6     0.05552  |                  .|*                  |
  7     0.03916  |                  .|*                  |
  8     0.08979  |                  .|**                 |
  9     0.03318  |                  .|*                  |
 10    -0.01774  |                  .|.                  |
 11     0.00435  |                  .|.                  |
 12     0.04693  |                  .|*                  |
 13     0.08131  |                  .|**                 |
 14     0.04345  |                  .|*                  |
 15     0.02521  |                  .|*                  |
```

REFERENCES

[1] C. W. Anderson, "Strategy Learning with Multilayer Connectionist Representations," *Proc. Fourth Int'l. Workshop on Machine Learning*, pp. 103-114, Morgan Kaufmann, June 1987.

[2] K. Hwang, W. J. Croft, G. H. Goble, B. W. Wah, F. A. Briggs, W. R. Simmons, and C. L. Coates, "A UNIX-based Local Computer Network with Load Balancing," *Computer*, vol. 15, no. 4, pp. 55-66, IEEE, April 1982.

[3] T. K. Miller III, R. S. Sutton, and P. J. Werbos (eds.), *Neural Networks for Control*, MIT Press, Cambridge, MA, 1990.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations By Error Propagation," pp. 318-362 in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, ed. D. E. Rumelhart, J. L McClelland and the PDP Research Group, MIT Press, Cambridge, MA, 1986.

[5] G. Tesauro and T. J. Sejnowski, "A Parallel Network that Learns to Play Backgammon," *Artificial Intelligence*, vol. 39, pp. 357-390, Elsevier Science Pub., 1989.